## 课程介绍

**1、讲师自我介绍**

（1）中华石杉

（2）BAT以及一线互联网公司出身，多年的大数据系统研发和架构经验，精通大数据技术栈，包括Hadoop，Spark，Storm，Hive，HBase，ZooKeeper，Elasticsearch，数据仓库，数据挖掘，大型Java系统架构，等等

（3）之前有过丰富的大型视频课程录制经验（spark），以及丰富企业内部培训和技术讲座、技术分享的经验

（4）Elastissearch技术，第一次合作

**2、讲课风格介绍**

（1）不喜欢ppt讲课，照本宣科，一本正经，对着文字和图片干讲 ----> txt文本边讲边写，画图软件手工画图

（2）不喜欢学术型的讲课，一板一眼，像老师在上课 ----> 像朋友之间聊天一样，口语化，轻松话，用大白话式的方法来讲解复杂的技术

（3）要录，就录最好的课程，知识体系全面；原理剖析透彻；全程实战演练驱动；搭配真实项目实战；讲解就业知识

**3、课程内容介绍**

\*\* （1）核心知识篇 \*\*

课程特点：

（1）使用最新Elasticsearch 5.2版本讲解，市面上的书籍和视频几乎都停留在2.x版本

（2）深入浅出ES核心工作原理，全部手工画图讲解，完全不同于市面上已有视频的PPT讲解

（3）涵盖Elasticsearch所有核心知识点，系统化，体系完整详细，有一定深度，包括完整Java开发示范

（3-1）全面的知识体系，包括了工作原理，文档管理，索引管理，搜索，聚合分析，分词，数据建模，Java API等知识

（3-2）知识足够深入和细节，完全秒杀市面上已有的书籍和视频，比如index segment merge原理，乐观锁并发控制，索引别名与零停机，相关度评分算法与定制，近似聚合算法，doc values与fielddata机制原理，父子关系数据建模，Java API执行scroll search等各种复杂操作，等等

（4）全程每讲必练，大量的案例实战和上机实验，实战出真知，实战中学知识，没有任何一讲是干讲ppt的

（5）包含一个实战项目，运用学到的知识，开发一个小型门户网站的搜索引擎和数据分析系统，运用ES几乎所有的核心知识，不像市面上的demo项目

（6）课程学完之后，学员可以掌握es所有核心知识点，理解es核心原理，而且能够熟练动手操作所有学到的知识和功能，并且能够掌握ES集群的基本部署，并且基于Java开发一个适用于中小型企业的搜索引擎以及数据分析系统，达到学完即可上手到中小型项目中使用的程度

\*\* （2）高手进阶篇 \*\*

课程特点：

（1）使用最新Elasticsearch 5.2版本讲解

（2）包含市面上几乎没有的所有Elasticsearch高级知识点：包含地理位置搜索与聚合分析，term vector，suggester search，搜索模板定制，query执行剖析，数十种最全面的聚合分析，span query，shard分配定制，es插件开发，等等，高级的知识点，这些知识点，市面上已有的书籍或视频几乎都没有

（3）全程每讲必练，大量的案例实战和上机实验

（4）包含一个复杂实战项目，运用学到的知识，开发一个复杂的基于地理位置的智能餐厅app的搜索引擎和数据分析系统，运用ES从核心篇到高级篇的所有高阶知识点

（5）课程学完之后，学员可以掌握es从核心到高阶的所有知识点，掌握完整的有深度的es知识体系，同时能够动手操作所有的知识点和功能，最后通过项目实战，能够在中小型公司中，基于Java开发一个可以基于地理位置进行搜索的高级搜索引擎，以及使用复杂聚合操作进行分析的高级实时数据分析系统

\*\* （3） 大型集群运维优化篇 \*\*

课程特点：

（1）最全面的Elasticsearch运维、管理、调优、故障处理的知识体系：企业级监控体系的搭建，企业级集群部署，集群日常管理策略，集群版本升级方案，集群基准压测方案，集群数据的备份和恢复，系统核心配置参数，性能调优方案，故障处理方案

（2）全程每讲必练，大量上机实验，所有的运维、管理、部署、优化，全部上机实验

（3）从零开始，逐步搭建出一个大型可扩展、高性能、监控体系完善、管理体系健全的分布式集群

（4）学完课程之后，学员除了可以开发复杂的es搜索/分析系统之外，还可以掌握在任何一个公司里，从零开始搭建一个分布式的大型es集群，并制定完善的监控，运维，管理，优化等方案

\*\* （4）大型项目架构篇 \*\*

课程特点：

（1）涵盖Elasticsearch目前最核心的两个应用领域，垂直搜索引擎，实时数据分析

（2）开发出2个企业级的大型复杂项目，是完全真实的大型企业项目，电商搜索引擎，电商实时数据分析平台

（2-1）大型电商搜索引擎，包括了真正复杂的大型企业，大型项目的商业级搜索引擎架构，包括了检索、数据更新、排序、分词、query分析等各个核心模块，同时架构上实现了复杂的缓存机制，热启动机制，防雪崩机制，自动降级高可用机制，等等

（2-2）大型电商实时数据分析平台，完整、复杂而且大型的电商数据分析，包括了完善的数据分析指标体系（运营指标，流量指标，销售转化指标，客户价值指标，商品指标，营销指标，风险控制指标，市场竞争指标），一站式构建出复杂的，企业级的，电商领域数据分析平台

（2-3）之所以要单独拉出一篇做大型项目实战，是因为，之前几篇讲的项目，多是架构较为简单，业务也不复杂的项目，主要适用于中小型公司，而且那两个项目主要是集中在运用ES的技术本身来开发出需要的功能来（搜索/分析）。这一篇讲解的项目，重点是采用大公司的大型复杂项目作为背景，让同学可以掌握基于ES技术的大型项目架构能力，达到架构师的水平。比如说大型电商搜索引擎，主要运用ES来实现，但是ES之外还有大型系统的复杂架构需要讲解。还有大型的电商实时数据分析平台，主要特点是业务繁琐而且复杂，需要基于ES构建出大型的数据分析平台架构。

（3）学完课程之后，学员在之前课程中掌握了es的企业级集群运维管理，复杂搜索引擎和数据分析引用开发的技术基础之上，现在可以运用所学知识，结合电商的领域知识，开发出业务真实而且复杂的，大型的搜索、分析等电商系统以及相关架构，彻底掌握运用ES和ELK相关技术栈在中大型企业中，开发大型项目架构的经验和能力

\*\*（5）ELK深入浅出篇 \*\*

课程特点：

（1）电商系统日志检索平台，采用ELK技术栈，会详细讲解logstash和kibana两个技术，包括logstash的插件机制，监控方案，大规模扩展方案，升级方案，性能调优方案，kibana的可视化展现方案，同时讲解如何使用ELK技术栈开发出大规模日志存储与检索平台

（2）最后，需要真正深入重点讲解logstash和kibana两门技术，并结合ES技术，采用ELK技术栈，实现大型企业级的日志采集和检索平台。

（3）学完课程后，学员应该可以彻底精通ELK技术栈，并能够使用ELK技术栈快速搭建日志检索平台，以及数据可视化平台

## 用大白话告诉你什么是 Elasticsearch

Elasticsearch，分布式，高性能，高可用，可伸缩的搜索和分析系统

1、什么是搜索？

2、如果用数据库做搜索会怎么样？

3、什么是全文检索、倒排索引和Lucene？

4、什么是Elasticsearch？

**1、什么是搜索？**

百度：我们比如说想找寻任何的信息的时候，就会上百度去搜索一下，比如说找一部自己喜欢的电影，或者说找一本喜欢的书，或者找一条感兴趣的新闻（提到搜索的第一印象）

百度 != 搜索，这是不对的

垂直搜索（站内搜索）

互联网的搜索：电商网站，招聘网站，新闻网站，各种app

IT系统的搜索：OA软件，办公自动化软件，会议管理，日程管理，项目管理，员工管理，搜索“张三”，“张三儿”，“张小三”；有个电商网站，卖家，后台管理系统，搜索“牙膏”，订单，“牙膏相关的订单”

搜索，就是在任何场景下，找寻你想要的信息，这个时候，会输入一段你要搜索的关键字，然后就期望找到这个关键字相关的有些信息

**2、如果用数据库做搜索会怎么样？**

做软件开发的话，或者对IT、计算机有一定的了解的话，都知道，数据都是存储在数据库里面的，比如说电商网站的商品信息，招聘网站的职位信息，新闻网站的新闻信息，等等吧。所以说，很自然的一点，如果说从技术的角度去考虑，如何实现如说，电商网站内部的搜索功能的话，就可以考虑，去使用数据库去进行搜索。

1、比方说，每条记录的指定字段的文本，可能会很长，比如说“商品描述”字段的长度，有长达数千个，甚至数万个字符，这个时候，每次都要对每条记录的所有文本进行扫描，懒判断说，你包不包含我指定的这个关键词（比如说“牙膏”）

2、还不能将搜索词拆分开来，尽可能去搜索更多的符合你的期望的结果，比如输入“生化机”，就搜索不出来“生化危机”

用数据库来实现搜索，是不太靠谱的。通常来说，性能会很差的。



**3、什么是全文检索和Lucene？**

（1）全文检索，倒排索引

（2）lucene，就是一个jar包，里面包含了封装好的各种建立倒排索引，以及进行搜索的代码，包括各种算法。我们就用java开发的时候，引入lucene jar，然后基于lucene的api进行去进行开发就可以了。用lucene，我们就可以去将已有的数据建立索引，lucene会在本地磁盘上面，给我们组织索引的数据结构。另外的话，我们也可以用lucene提供的一些功能和api来针对磁盘上额

**4、什么是Elasticsearch？**

## Elasticsearch 的功能、适用场景以及特点介绍

1、Elasticsearch的功能，干什么的

2、Elasticsearch的适用场景，能在什么地方发挥作用

3、Elasticsearch的特点，跟其他类似的东西不同的地方在哪里

**1、Elasticsearch的功能**

（1）分布式的搜索引擎和数据分析引擎

搜索：百度，网站的站内搜索，IT系统的检索

数据分析：电商网站，最近7天牙膏这种商品销量排名前10的商家有哪些；新闻网站，最近1个月访问量排名前3的新闻版块是哪些

分布式，搜索，数据分析

（2）全文检索，结构化检索，数据分析

全文检索：我想搜索商品名称包含牙膏的商品，select \* from products where product\_name like "%牙膏%"

结构化检索：我想搜索商品分类为日化用品的商品都有哪些，select \* from products where category\_id='日化用品'

部分匹配、自动完成、搜索纠错、搜索推荐

数据分析：我们分析每一个商品分类下有多少个商品，select category\_id,count(\*) from products group by category\_id

（3）对海量数据进行近实时的处理

分布式：ES自动可以将海量数据分散到多台服务器上去存储和检索

海联数据的处理：分布式以后，就可以采用大量的服务器去存储和检索数据，自然而然就可以实现海量数据的处理了

近实时：检索个数据要花费1小时（这就不要近实时，离线批处理，batch-processing）；在秒级别对数据进行搜索和分析

跟分布式/海量数据相反的：lucene，单机应用，只能在单台服务器上使用，最多只能处理单台服务器可以处理的数据量

**2、Elasticsearch的适用场景**

国外

（1）维基百科，类似百度百科，牙膏，牙膏的维基百科，全文检索，高亮，搜索推荐

（2）The Guardian（国外新闻网站），类似搜狐新闻，用户行为日志（点击，浏览，收藏，评论）+社交网络数据（对某某新闻的相关看法），数据分析，给到每篇新闻文章的作者，让他知道他的文章的公众反馈（好，坏，热门，垃圾，鄙视，崇拜）

（3）Stack Overflow（国外的程序异常讨论论坛），IT问题，程序的报错，提交上去，有人会跟你讨论和回答，全文检索，搜索相关问题和答案，程序报错了，就会将报错信息粘贴到里面去，搜索有没有对应的答案

（4）GitHub（开源代码管理），搜索上千亿行代码

（5）电商网站，检索商品

（6）日志数据分析，logstash采集日志，ES进行复杂的数据分析（ELK技术，elasticsearch+logstash+kibana）

（7）商品价格监控网站，用户设定某商品的价格阈值，当低于该阈值的时候，发送通知消息给用户，比如说订阅牙膏的监控，如果高露洁牙膏的家庭套装低于50块钱，就通知我，我就去买

（8）BI系统，商业智能，Business Intelligence。比如说有个大型商场集团，BI，分析一下某某区域最近3年的用户消费金额的趋势以及用户群体的组成构成，产出相关的数张报表，\*\*区，最近3年，每年消费金额呈现100%的增长，而且用户群体85%是高级白领，开一个新商场。ES执行数据分析和挖掘，Kibana进行数据可视化

国内

（9）国内：站内搜索（电商，招聘，门户，等等），IT系统搜索（OA，CRM，ERP，等等），数据分析（ES热门的一个使用场景）

**3、Elasticsearch的特点**

（1）可以作为一个大型分布式集群（数百台服务器）技术，处理PB级数据，服务大公司；也可以运行在单机上，服务小公司

（2）Elasticsearch不是什么新技术，主要是将全文检索、数据分析以及分布式技术，合并在了一起，才形成了独一无二的ES；lucene（全文检索），商用的数据分析软件（也是有的），分布式数据库（mycat）

（3）对用户而言，是开箱即用的，非常简单，作为中小型的应用，直接3分钟部署一下ES，就可以作为生产环境的系统来使用了，数据量不大，操作不是太复杂

（4）数据库的功能面对很多领域是不够用的（事务，还有各种联机事务型的操作）；特殊的功能，比如全文检索，同义词处理，相关度排名，复杂数据分析，海量数据的近实时处理；Elasticsearch作为传统数据库的一个补充，提供了数据库所不不能提供的很多功能

## 手工画图剖析 Elasticsearch 核心概念

核心概念：NRT、索引、分片、副本等。

1、lucene和elasticsearch的前世今生

2、elasticsearch的核心概念

3、elasticsearch核心概念 vs. 数据库核心概念

**1、lucene和elasticsearch的前世今生**

lucene，最先进、功能最强大的搜索库，直接基于lucene开发，非常复杂，api复杂（实现一些简单的功能，写大量的java代码），需要深入理解原理（各种索引结构）

elasticsearch，基于lucene，隐藏复杂性，提供简单易用的restful api接口、java api接口（还有其他语言的api接口）

（1）分布式的文档存储引擎

（2）分布式的搜索引擎和分析引擎

（3）分布式，支持PB级数据

开箱即用，优秀的默认参数，不需要任何额外设置，完全开源

关于elasticsearch的一个传说，有一个程序员失业了，陪着自己老婆去英国伦敦学习厨师课程。程序员在失业期间想给老婆写一个菜谱搜索引擎，觉得lucene实在太复杂了，就开发了一个封装了lucene的开源项目，compass。后来程序员找到了工作，是做分布式的高性能项目的，觉得compass不够，就写了elasticsearch，让lucene变成分布式的系统。

**2、elasticsearch的核心概念**

（1）Near Realtime（NRT）：近实时，两个意思，从写入数据到数据可以被搜索到有一个小延迟（大概1秒）；基于es执行搜索和分析可以达到秒级



（2）Cluster：集群，包含多个节点，每个节点属于哪个集群是通过一个配置（集群名称，默认是elasticsearch）来决定的，对于中小型应用来说，刚开始一个集群就一个节点很正常

（3）Node：节点，集群中的一个节点，节点也有一个名称（默认是随机分配的），节点名称很重要（在执行运维管理操作的时候），默认节点会去加入一个名称为“elasticsearch”的集群，如果直接启动一堆节点，那么它们会自动组成一个elasticsearch集群，当然一个节点也可以组成一个elasticsearch集群

（4）Document&field：文档，es中的最小数据单元，一个document可以是一条客户数据，一条商品分类数据，一条订单数据，通常用JSON数据结构表示，每个index下的type中，都可以去存储多个document。一个document里面有多个field，每个field就是一个数据字段。

product document

{

"product\_id": "1",

"product\_name": "高露洁牙膏",

"product\_desc": "高效美白",

"category\_id": "2",

"category\_name": "日化用品"

}

（5）Index：索引，包含一堆有相似结构的文档数据，比如可以有一个客户索引，商品分类索引，订单索引，索引有一个名称。一个index包含很多document，一个index就代表了一类类似的或者相同的document。比如说建立一个product index，商品索引，里面可能就存放了所有的商品数据，所有的商品document。

（6）Type：类型，每个索引里都可以有一个或多个type，type是index中的一个逻辑数据分类，一个type下的document，都有相同的field，比如博客系统，有一个索引，可以定义用户数据type，博客数据type，评论数据type。

商品index，里面存放了所有的商品数据，商品document

但是商品分很多种类，每个种类的document的field可能不太一样，比如说电器商品，可能还包含一些诸如售后时间范围这样的特殊field；生鲜商品，还包含一些诸如生鲜保质期之类的特殊field

type，日化商品type，电器商品type，生鲜商品type

日化商品type：product\_id，product\_name，product\_desc，category\_id，category\_name

电器商品type：product\_id，product\_name，product\_desc，category\_id，category\_name，service\_period

生鲜商品type：product\_id，product\_name，product\_desc，category\_id，category\_name，eat\_period

每一个type里面，都会包含一堆document

{

"product\_id": "2",

"product\_name": "长虹电视机",

"product\_desc": "4k高清",

"category\_id": "3",

"category\_name": "电器",

"service\_period": "1年"

}

{

"product\_id": "3",

"product\_name": "基围虾",

"product\_desc": "纯天然，冰岛产",

"category\_id": "4",

"category\_name": "生鲜",

"eat\_period": "7天"

}

（7）shard：单台机器无法存储大量数据，es可以将一个索引中的数据切分为多个shard，分布在多台服务器上存储。有了shard就可以横向扩展，存储更多数据，让搜索和分析等操作分布到多台服务器上去执行，提升吞吐量和性能。每个shard都是一个lucene index。

（8）replica：任何一个服务器随时可能故障或宕机，此时shard可能就会丢失，因此可以为每个shard创建多个replica副本。replica可以在shard故障时提供备用服务，保证数据不丢失，多个replica还可以提升搜索操作的吞吐量和性能。primary shard（建立索引时一次设置，不能修改，默认5个），replica shard（随时修改数量，默认1个），默认每个索引10个shard，5个primary shard，5个replica shard，最小的高可用配置，是2台服务器。



**3、elasticsearch核心概念 vs. 数据库核心概念**

Elasticsearch 数据库

-----------------------------------------

Document 行

Type 表

Index 库

## 在windows 上安装和启动 Elasticsearch

1、安装JDK，至少1.8.0\_73以上版本，java -version

2、下载和解压缩Elasticsearch安装包，目录结构

3、启动Elasticsearch：bin\elasticsearch.bat，es本身特点之一就是开箱即用，如果是中小型应用，数据量少，操作不是很复杂，直接启动就可以用了

4、检查ES是否启动成功：http://localhost:9200/?pretty

name: node名称

cluster\_name: 集群名称（默认的集群名称就是elasticsearch）

version.number: 5.2.0，es版本号

{

"name" : "4onsTYV",

"cluster\_name" : "elasticsearch",

"cluster\_uuid" : "nKZ9VK\_vQdSQ1J0Dx9gx1Q",

"version" : {

"number" : "5.2.0",

"build\_hash" : "24e05b9",

"build\_date" : "2017-01-24T19:52:35.800Z",

"build\_snapshot" : false,

"lucene\_version" : "6.4.0"

},

"tagline" : "You Know, for Search"

}

5、修改集群名称：elasticsearch.yml

6、下载和解压缩Kibana安装包，使用里面的开发界面，去操作elasticsearch，作为我们学习es知识点的一个主要的界面入口

7、启动Kibana：bin\kibana.bat

8、进入Dev Tools界面

9、GET \_cluster/health

## 快速入门-电商案例一

电商的商品管理：集群健康检查以及文档的 CRUD

1、document数据格式

2、电商网站商品管理案例：背景介绍

3、简单的集群管理

4、商品的CRUD操作（document CRUD操作）

**1、document数据格式**

面向文档的搜索分析引擎

（1）应用系统的数据结构都是面向对象的，复杂的

（2）对象数据存储到数据库中，只能拆解开来，变为扁平的多张表，每次查询的时候还得还原回对象格式，相当麻烦

（3）ES是面向文档的，文档中存储的数据结构，与面向对象的数据结构是一样的，基于这种文档数据结构，es可以提供复杂的索引，全文检索，分析聚合等功能

（4）es的document用json数据格式来表达

public class Employee {

private String email;

private String firstName;

private String lastName;

private EmployeeInfo info;

private Date joinDate;

}

private class EmployeeInfo {

private String bio; // 性格

private Integer age;

private String[] interests; // 兴趣爱好

}

EmployeeInfo info = new EmployeeInfo();

info.setBio("curious and modest");

info.setAge(30);

info.setInterests(new String[]{"bike", "climb"});

Employee employee = new Employee();

employee.setEmail("zhangsan@sina.com");

employee.setFirstName("san");

employee.setLastName("zhang");

employee.setInfo(info);

employee.setJoinDate(new Date());

employee对象：里面包含了Employee类自己的属性，还有一个EmployeeInfo对象

两张表：employee表，employee\_info表，将employee对象的数据重新拆开来，变成Employee数据和EmployeeInfo数据

employee表：email，first\_name，last\_name，join\_date，4个字段

employee\_info表：bio，age，interests，3个字段；此外还有一个外键字段，比如employee\_id，关联着employee表

{

"email": "zhangsan@sina.com",

"first\_name": "san",

"last\_name": "zhang",

"info": {

"bio": "curious and modest",

"age": 30,

"interests": [ "bike", "climb" ]

},

"join\_date": "2017/01/01"

}

我们就明白了es的document数据格式和数据库的关系型数据格式的区别

**2、电商网站商品管理案例背景介绍**

有一个电商网站，需要为其基于ES构建一个后台系统，提供以下功能：

（1）对商品信息进行CRUD（增删改查）操作

（2）执行简单的结构化查询

（3）可以执行简单的全文检索，以及复杂的phrase（短语）检索

（4）对于全文检索的结果，可以进行高亮显示

（5）对数据进行简单的聚合分析

**3、简单的集群管理**

（1）快速检查集群的健康状况

es提供了一套api，叫做cat api，可以查看es中各种各样的数据

GET /\_cat/health?v

epoch timestamp cluster status node.total node.data shards pri relo init unassign pending\_tasks max\_task\_wait\_time active\_shards\_percent

1488006741 15:12:21 elasticsearch yellow 1 1 1 1 0 0 1 0 - 50.0%

epoch timestamp cluster status node.total node.data shards pri relo init unassign pending\_tasks max\_task\_wait\_time active\_shards\_percent

1488007113 15:18:33 elasticsearch green 2 2 2 1 0 0 0 0 - 100.0%

epoch timestamp cluster status node.total node.data shards pri relo init unassign pending\_tasks max\_task\_wait\_time active\_shards\_percent

1488007216 15:20:16 elasticsearch yellow 1 1 1 1 0 0 1 0 - 50.0%

如何快速了解集群的健康状况？green、yellow、red？

1. green：每个索引的primary shard和replica shard都是active状态的

2. yellow：每个索引的primary shard都是active状态的，但是部分replica shard不是active状态，处于不可用的状态

3. red：不是所有索引的primary shard都是active状态的，部分索引有数据丢失了

为什么现在会处于一个yellow状态？

我们现在就一个笔记本电脑，就启动了一个es进程，相当于就只有一个node。现在es中有一个index，就是kibana自己内置建立的index。由于默认的配置是给每个index分配5个primary shard和5个replica shard，而且primary shard和replica shard不能在同一台机器上（为了容错）。现在kibana自己建立的index是1个primary shard和1个replica shard。当前就一个node，所以只有1个primary shard被分配了和启动了，但是一个replica shard没有第二台机器去启动。

做一个小实验：此时只要启动第二个es进程，就会在es集群中有2个node，然后那1个replica shard就会自动分配过去，然后cluster status就会变成green状态。

（2）快速查看集群中有哪些索引

GET /\_cat/indices?v

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open .kibana rUm9n9wMRQCCrRDEhqneBg 1 1 1 0 3.1kb 3.1kb

（3）简单的索引操作

创建索引：PUT /test\_index?pretty

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open test\_index XmS9DTAtSkSZSwWhhGEKkQ 5 1 0 0 650b 650b

yellow open .kibana rUm9n9wMRQCCrRDEhqneBg 1 1 1 0 3.1kb 3.1kb

删除索引：DELETE /test\_index?pretty

health status index uuid pri rep docs.count docs.deleted store.size pri.store.size

yellow open .kibana rUm9n9wMRQCCrRDEhqneBg 1 1 1 0 3.1kb 3.1kb

**4、商品的CRUD操作**

PS：目前这里还没有

（1）新增商品：新增文档，建立索引

语法：

PUT /index/type/id

{

"json数据"

}

eg：

PUT /ecommerce/product/1

{

"name" : "gaolujie yagao",

"desc" : "gaoxiao meibai",

"price" : 30,

"producer" : "gaolujie producer",

"tags": [ "meibai", "fangzhu" ]

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 1,

"result": "created",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"created": true

}

// 写一个 primary shard，然后 replica shard 也需要有这个数据。一个节点所以 succesful 是 1， 两个 shard 所以\_shards的total值是 2

PUT /ecommerce/product/2

{

"name" : "jiajieshi yagao",

"desc" : "youxiao fangzhu",

"price" : 25,

"producer" : "jiajieshi producer",

"tags": [ "fangzhu" ]

}

PUT /ecommerce/product/3

{

"name" : "zhonghua yagao",

"desc" : "caoben zhiwu",

"price" : 40,

"producer" : "zhonghua producer",

"tags": [ "qingxin" ]

}

es会自动建立index和type，不需要提前创建，而且es默认会对document每个field都建立倒排索引，让其可以被搜索

（2）查询商品：检索文档

语法：GET /{index}/{type}/{id}

GET /ecommerce/product/1

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 1,

"found": true,

"\_source": {

"name": "gaolujie yagao",

"desc": "gaoxiao meibai",

"price": 30,

"producer": "gaolujie producer",

"tags": [

"meibai",

"fangzhu"

]

}

}

（3）修改商品：替换文档

PUT /ecommerce/product/1

{

"name" : "jiaqiangban gaolujie yagao",

"desc" : "gaoxiao meibai",

"price" : 30,

"producer" : "gaolujie producer",

"tags": [ "meibai", "fangzhu" ]

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 1,

"result": "created",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"created": true

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 2,

"result": "updated",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"created": false

}

PUT /ecommerce/product/1

{

"name" : "jiaqiangban gaolujie yagao"

}

替换方式有一个不好，即使必须带上所有的field，才能去进行信息的修改

（4）修改商品：更新文档

POST /ecommerce/product/1/\_update

{

"doc": {

"name": "jiaqiangban gaolujie yagao"

}

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 8,

"result": "updated",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

}

}

我的风格，其实有选择的情况下，不太喜欢念ppt，或者照着文档做，或者直接粘贴写好的代码，尽量是纯手敲代码

（5）删除商品：删除文档

DELETE /ecommerce/product/1

{

"found": true,

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"\_version": 9,

"result": "deleted",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

}

}

{

"\_index": "ecommerce",

"\_type": "product",

"\_id": "1",

"found": false

}

## 快速入门-电商案例二

主要是对查询的一些语法的体验

1、query string search

2、query DSL

3、query filter

4、full-text search

5、phrase search

6、highlight search

题外话：把英文翻译成中文，让我觉得很别扭，term翻译成词项

**1、query string search**

搜索全部商品：

GET /ecommerce/product/\_search

took：耗费了几毫秒

timed\_out：是否超时，这里是没有

\_shards：数据拆成了5个分片，所以对于搜索请求，会打到所有的primary shard（或者是它的某个replica shard也可以）

hits.total：查询结果的数量，3个document

hits.max\_score：score的含义，就是document对于一个search的相关度的匹配分数，越相关，就越匹配，分数也高

hits.hits：包含了匹配搜索的document的详细数据

|  |
| --- |
| {  "took": 2,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "failed": 0  },  "hits": {  "total": 3,  "max\_score": 1,  "hits": [  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "2",  "\_score": 1,  "\_source": {  "name": "jiajieshi yagao",  "desc": "youxiao fangzhu",  "price": 25,  "producer": "jiajieshi producer",  "tags": [  "fangzhu"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "1",  "\_score": 1,  "\_source": {  "name": "gaolujie yagao",  "desc": "gaoxiao meibai",  "price": 30,  "producer": "gaolujie producer",  "tags": [  "meibai",  "fangzhu"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "3",  "\_score": 1,  "\_source": {  "name": "zhonghua yagao",  "desc": "caoben zhiwu",  "price": 40,  "producer": "zhonghua producer",  "tags": [  "qingxin"  ]  }  }  ]  }  } |

query string search的由来，因为search参数都是以http请求的query string来附带的

搜索商品名称中包含yagao的商品，而且按照售价降序排序：GET /ecommerce/product/\_search?q=name:yagao&sort=price:desc

适用于临时的在命令行使用一些工具，比如curl，快速的发出请求，来检索想要的信息；但是如果查询请求很复杂，是很难去构建的

在生产环境中，几乎很少使用query string search

**2、query DSL**

DSL：Domain Specified Language，特定领域的语言

http request body：请求体，可以用json的格式来构建查询语法，比较方便，可以构建各种复杂的语法，比query string search肯定强大多了

查询所有的商品

GET /ecommerce/product/\_search

{

"query": { "match\_all": {} }

}

查询名称包含yagao的商品，同时按照价格降序排序

GET /ecommerce/product/\_search

{

"query" : {

"match" : {

"name" : "yagao"

}

},

"sort": [

{ "price": "desc" }

]

}

分页查询商品，总共3条商品，假设每页就显示1条商品，现在显示第2页，所以就查出来第2个商品

GET /ecommerce/product/\_search

{

"query": { "match\_all": {} },

"from": 1,

"size": 1

}

指定要查询出来商品的名称和价格就可以

GET /ecommerce/product/\_search

{

"query": { "match\_all": {} },

"\_source": ["name", "price"]

}

更加适合生产环境的使用，可以构建复杂的查询

**3、query filter**

搜索商品名称包含yagao，而且售价大于25元的商品

GET /ecommerce/product/\_search

{

"query" : {

"bool" : {

"must" : {

"match" : {

"name" : "yagao"

}

},

"filter" : {

"range" : {

"price" : { "gt" : 25 }

}

}

}

}

}

**4、full-text search（全文检索）**

GET /ecommerce/product/\_search

{

"query" : {

"match" : {

"producer" : "yagao producer"

}

}

}

PS：尽量，无论是学什么技术，比如说你当初学java，学linux，学shell，学javascript，学hadoop。。。。一定自己动手，特别是手工敲各种命令和代码，切记切记，减少复制粘贴的操作。只有自己动手手工敲，学习效果才最好。

**过程：**producer这个字段，会先被拆解，建立倒排索引如下所示

special 4

yagao 4

producer 1,2,3,4

gaolujie 1

zhognhua 3

jiajieshi 2

搜索的单词yagao producer 会被拆解为 yagao和producer

结果：

|  |
| --- |
| {  "took": 4,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "failed": 0  },  "hits": {  "total": 4,  "max\_score": 0.70293105,  "hits": [  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "4",  "\_score": 0.70293105,  "\_source": {  "name": "special yagao",  "desc": "special meibai",  "price": 50,  "producer": "special yagao producer",  "tags": [  "meibai"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "1",  "\_score": 0.25811607,  "\_source": {  "name": "gaolujie yagao",  "desc": "gaoxiao meibai",  "price": 30,  "producer": "gaolujie producer",  "tags": [  "meibai",  "fangzhu"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "3",  "\_score": 0.25811607,  "\_source": {  "name": "zhonghua yagao",  "desc": "caoben zhiwu",  "price": 40,  "producer": "zhonghua producer",  "tags": [  "qingxin"  ]  }  },  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "2",  "\_score": 0.1805489,  "\_source": {  "name": "jiajieshi yagao",  "desc": "youxiao fangzhu",  "price": 25,  "producer": "jiajieshi producer",  "tags": [  "fangzhu"  ]  }  }  ]  }  } |

**5、phrase search（短语搜索）**

跟全文检索相对应，相反，全文检索会将输入的搜索串拆解开来，去倒排索引里面去一一匹配，只要能匹配上任意一个拆解后的单词，就可以作为结果返回

phrase search，要求输入的搜索串，必须在指定的字段文本中，完全包含一模一样的，才可以算匹配，才能作为结果返回

GET /ecommerce/product/\_search

{

"query" : {

"match\_phrase" : {

"producer" : "yagao producer"

}

}

}

结果：

|  |
| --- |
| {  "took": 11,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "failed": 0  },  "hits": {  "total": 1,  "max\_score": 0.70293105,  "hits": [  {  "\_index": "ecommerce",  "\_type": "product",  "\_id": "4",  "\_score": 0.70293105,  "\_source": {  "name": "special yagao",  "desc": "special meibai",  "price": 50,  "producer": "special yagao producer",  "tags": [  "meibai"  ]  }  }  ]  }  } |

**6、highlight search（高亮搜索结果）**

GET /ecommerce/product/\_search

{

"query" : {

"match" : {

"producer" : "producer"

}

},

"highlight": {

"fields" : {

"producer" : {}

}

}

}

## 快速入门-电商案例三

主要是体验对分析方面的工程

**第一个分析需求：**

计算每个tag下的商品数量

GET /ecommerce/product/\_search

{

"aggs": {

"group\_by\_tags": {

"terms": { "field": "tags" }

}

}

}

上面执行后就会报异常。

fileddata的默认值是禁止的，如果需要聚合，就需要设置为 true，

因此将文本field的fielddata属性设置为true

PUT /ecommerce/\_mapping/product

{

"properties": {

"tags": {

"type": "text",

"fielddata": true

}

}

}

GET /ecommerce/product/\_search

{

"size": 0,

"aggs": {

"all\_tags": {

"terms": { "field": "tags" }

}

}

}

结果：

|  |
| --- |
| {  "took": 20,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "failed": 0  },  "hits": {  "total": 4,  "max\_score": 0,  "hits": []  },  "aggregations": {  "group\_by\_tags": {  "doc\_count\_error\_upper\_bound": 0,  "sum\_other\_doc\_count": 0,  "buckets": [  {  "key": "fangzhu",  "doc\_count": 2  },  {  "key": "meibai",  "doc\_count": 2  },  {  "key": "qingxin",  "doc\_count": 1  }  ]  }  }  } |

**第二个聚合分析的需求：**

对名称中包含yagao的商品，计算每个tag下的商品数量

GET /ecommerce/product/\_search

{

"size": 0,

"query": {

"match": {

"name": "yagao"

}

},

"aggs": {

"all\_tags": {

"terms": {

"field": "tags"

}

}

}

}

**第三个聚合分析的需求：**

先分组，再算每组的平均值，计算每个tag下的商品的平均价格

GET /ecommerce/product/\_search

{

"size": 0,

"aggs" : {

"group\_by\_tags" : {

"terms" : { "field" : "tags" },

"aggs" : {

"avg\_price" : {

"avg" : { "field" : "price" }

}

}

}

}

}

|  |
| --- |
| {  "took": 8,  "timed\_out": false,  "\_shards": {  "total": 5,  "successful": 5,  "failed": 0  },  "hits": {  "total": 4,  "max\_score": 0,  "hits": []  },  "aggregations": {  "group\_by\_tags": {  "doc\_count\_error\_upper\_bound": 0,  "sum\_other\_doc\_count": 0,  "buckets": [  {  "key": "fangzhu",  "doc\_count": 2,  "avg\_price": {  "value": 27.5  }  },  {  "key": "meibai",  "doc\_count": 2,  "avg\_price": {  "value": 40  }  },  {  "key": "qingxin",  "doc\_count": 1,  "avg\_price": {  "value": 40  }  }  ]  }  }  } |

**第四个数据分析需求：**

计算每个tag下的商品的平均价格，并且按照平均价格降序排序

GET /ecommerce/product/\_search

{

"size": 0,

"aggs" : {

"all\_tags" : {

"terms" : { "field" : "tags", "order": { "avg\_price": "desc" } },

"aggs" : {

"avg\_price" : {

"avg" : { "field" : "price" }

}

}

}

}

}

我们现在全部都是用es的restful api在学习和讲解es的所欲知识点和功能点，但是没有使用一些编程语言去讲解（比如java），原因有以下：

1、es最重要的api，让我们进行各种尝试、学习甚至在某些环境下进行使用的api，就是restful api。如果你学习不用es restful api，比如我上来就用java api来讲es，也是可以的，但是你根本就漏掉了es知识的一大块，你都不知道它最重要的restful api是怎么用的

2、讲知识点，用es restful api，更加方便，快捷，不用每次都写大量的java代码，能加快讲课的效率和速度，更加易于同学们关注es本身的知识和功能的学习

3、我们通常会讲完es知识点后，开始详细讲解java api，如何用java api执行各种操作

4、我们每个篇章都会搭配一个项目实战，项目实战是完全基于java去开发的真实项目和系统

**第五个数据分析需求：**

按照指定的价格范围区间进行分组，然后在每组内再按照tag进行分组，最后再计算每组的平均价格

GET /ecommerce/product/\_search

{

"size": 0,

"aggs": {

"group\_by\_price": {

"range": {

"field": "price",

"ranges": [

{

"from": 0,

"to": 20

},

{

"from": 20,

"to": 40

},

{

"from": 40,

"to": 50

}

]

},

"aggs": {

"group\_by\_tags": {

"terms": {

"field": "tags"

},

"aggs": {

"average\_price": {

"avg": {

"field": "price"

}

}

}

}

}

}

}

}

## Elasticsearch 的基础分布式架构

1、Elasticsearch对复杂分布式机制的透明隐藏特性

2、Elasticsearch的垂直扩容与水平扩容

3、增减或减少节点时的数据rebalance

4、master节点

5、节点对等的分布式架构



**1、Elasticsearch对复杂分布式机制的透明隐藏特性**

Elasticsearch是一套分布式的系统，分布式是为了应对大数据量

隐藏了复杂的分布式机制

首先：分片机制（我们之前随随便便就将一些document插入到es集群中去了，我们有没有care过数据怎么进行分片的，数据到哪个shard中去）

cluster discovery（集群发现机制，我们之前在做那个集群status从yellow转green的实验里，直接启动了第二个es进程，那个进程作为一个node自动就发现了集群，并且加入了进去，还接受了部分数据，replica shard）

shard负载均衡（举例，假设现在有3个节点，总共有25个shard要分配到3个节点上去，es会自动进行均匀分配，以保持每个节点的均衡的读写负载请求）

shard副本，请求路由，集群扩容，shard重分配

**2、Elasticsearch的垂直扩容与水平扩容**

垂直扩容：采购更强大的服务器，成本非常高昂，而且会有瓶颈，假设世界上最强大的服务器容量就是10T，但是当你的总数据量达到5000T的时候，你要采购多少台最强大的服务器啊

**水平扩容**：业界经常采用的方案，采购越来越多的普通服务器，性能比较一般，但是很多普通服务器组织在一起，就能构成强大的计算和存储能力

普通服务器：1T，1万，100万

强大服务器：10T，50万，500万

扩容对应用程序的透明性

**3、增减或减少节点时的数据rebalance**

保持负载均衡

**4、master节点**

（1）创建或删除索引

（2）增加或删除节点

**5、节点平等的分布式架构**

（1）节点对等，每个节点都能接收所有的请求

（2）自动请求路由

（3）响应收集

## shard&replica 机制再次梳理以及单 node 环境中创建 index 图解

1、shard&replica机制再次梳理

2、图解单node环境下创建index是什么样子的

**1、shard&replica机制再次梳理**

（1）index包含多个shard

（2）每个shard都是一个最小工作单元，承载部分数据，lucene实例，完整的建立索引和处理请求的能力，一个 Document 只会在一个 shard 里面。

（3）增减节点时，shard会自动在nodes中负载均衡

（4）primary shard和replica shard，每个document肯定只存在于某一个primary shard以及其对应的replica shard中，不可能存在于多个primary shard

（5）replica shard是primary shard的副本，负责容错，以及承担读请求负载

（6）primary shard的数量在创建索引的时候就固定了，replica shard的数量可以随时修改

（7）primary shard的默认数量是5，replica默认是1，默认有10个shard，5个primary shard，5个replica shard

（8）primary shard 不能和自己的replica shard放在同一个节点上（否则节点宕机，primary shard和副本都丢失，起不到容错的作用），但是可以和其他primary shard的replica shard放在同一个节点上

**2、图解单node环境下创建index是什么样子的**

（1）单node环境下，创建一个index，有3个primary shard，3个replica shard

（2）集群status是yellow

（3）这个时候，只会将3个primary shard分配到仅有的一个node上去，另外3个replica shard是无法分配的

（4）集群可以正常工作，但是一旦出现节点宕机，数据全部丢失，而且集群不可用，无法承接任何请求

PUT /test\_index

{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

}

}



## 图解 2 个 node 环境下 replica shard 是如何分配的

1、图解2个node环境下replica shard是如何分配的

其实重点就是，如果是多个服务器的话，replica shard 就会被分配。根据 primary shard 和 replica shard 不能再一台服务器上的原则。

（1）replica shard分配：3个primary shard，3个replica shard，1 node

（2）primary ---> replica同步

（3）读请求：primary/replica



## 图解横向扩容过程，如何超出扩容极限，以及如何提升容错性

（1）primary&replica自动负载均衡，6个shard，3 primary，3 replica

（2）每个node有更少的shard，IO/CPU/Memory资源给每个shard分配更多，每个shard性能更好

（3）扩容的极限，6个shard（3 primary，3 replica），最多扩容到6台机器，每个shard可以占用单台服务器的所有资源，性能最好

（4）超出扩容极限，动态修改replica数量，9个shard（3primary，6 replica），扩容到9台机器，比3台机器时，拥有3倍的读吞吐量

（5）3台机器下，9个shard（3 primary，6 replica），资源更少，但是容错性更好，最多容纳2台机器宕机，6个shard只能容纳0台机器宕机

（6）这里的这些知识点，你综合起来看，就是说，一方面告诉你扩容的原理，怎么扩容，怎么提升系统整体吞吐量；另一方面要考虑到系统的容错性，怎么保证提高容错性，让尽可能多的服务器宕机，保证数据不丢失



## 图解 Elasticsearch 容错机制：master 选举，replica 容错，数据恢复

（1）9 shard，3 node

（2）master node宕机，自动master选举，red

（3）replica容错：新master将replica提升为primary shard，yellow

（4）重启宕机node，master copy replica到该node，使用原有的shard并同步宕机后的修改，green

## 初步解析 document 的核心元数据以及图解剖析 index 创建反例

1、\_index元数据

2、\_type元数据

3、\_id元数据

**1、\_index元数据**

（1）代表一个document存放在哪个index中

（2）类似的数据放在一个索引，非类似的数据放不同索引：product index（包含了所有的商品），sales index（包含了所有的商品销售数据），inventory index（包含了所有库存相关的数据）。如果你把比如product，sales，human resource（employee），全都放在一个大的index里面，比如说company index，不合适的。

（3）index中包含了很多类似的document：类似是什么意思，其实指的就是说，这些document的fields很大一部分是相同的，你说你放了3个document，每个document的fields都完全不一样，这就不是类似了，就不太适合放到一个index里面去了。

（4）索引名称必须是小写的，不能用下划线开头，不能包含逗号：product，website，blog

**2、\_type元数据**

（1）代表document属于index中的哪个类别（type）

（2）一个索引通常会划分为多个type，逻辑上对index中有些许不同的几类数据进行分类：因为一批相同的数据，可能有很多相同的fields，但是还是可能会有一些轻微的不同，可能会有少数fields是不一样的，举个例子，就比如说，商品，可能划分为电子商品，生鲜商品，日化商品，等等。

（3）type名称可以是大写或者小写，但是同时不能用下划线开头，不能包含逗号

**3、\_id元数据**

（1）代表document的唯一标识，与index和type一起，可以唯一标识和定位一个document

（2）我们可以手动指定document的id（put /index/type/id），也可以不指定，由es自动为我们创建一个id

## document id 的手动执行与自动生成两种方式解析

1、手动指定document id

2、自动生成document id

**1、手动指定document id**

（1）根据应用情况来说，是否满足手动指定document id的前提：

一般来说，是从某些其他的系统中，导入一些数据到es时，会采取这种方式，就是使用系统中已有数据的唯一标识，作为es中document的id。举个例子，比如说，我们现在在开发一个电商网站，做搜索功能，或者是OA系统，做员工检索功能。这个时候，数据首先会在网站系统或者IT系统内部的数据库中，会先有一份，此时就肯定会有一个数据库的primary key（自增长，UUID，或者是业务编号）。如果将数据导入到es中，此时就比较适合采用数据在数据库中已有的primary key。

如果说，我们是在做一个系统，这个系统主要的数据存储就是es一种，也就是说，数据产生出来以后，可能就没有id，直接就放es一个存储，那么这个时候，可能就不太适合说手动指定document id的形式了，因为你也不知道id应该是什么，此时可以采取下面要讲解的让es自动生成id的方式。

（2）put /index/type/id

PUT /test\_index/test\_type/2

{

"test\_content": "my test"

}

**2、自动生成document id**

（1）post /index/type

POST /test\_index/test\_type

{

"test\_content": "my test"

}

{

"\_index": "test\_index",

"\_type": "test\_type",

"\_id": "AVp4RN0bhjxldOOnBxaE",

"\_version": 1,

"result": "created",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"created": true

}

（2）自动生成的id，长度为20个字符，URL安全，base64编码，GUID，分布式系统并行生成时不可能会发生冲突

## document 的\_source元数据以及定制返回结果解析

**1、\_source元数据**

put /test\_index/test\_type/1

{

"test\_field1": "test field1",

"test\_field2": "test field2"

}

get /test\_index/test\_type/1

{

"\_index": "test\_index",

"\_type": "test\_type",

"\_id": "1",

"\_version": 2,

"found": true,

"\_source": {

"test\_field1": "test field1",

"test\_field2": "test field2"

}

}

\_source元数据：就是说，我们在创建一个document的时候，使用的那个放在request body中的json串，默认情况下，在get的时候，会原封不动的给我们返回回来。

------------------------------------------------------------------------------------------------------------------

**2、定制返回结果**

定制返回的结果，指定\_source中，返回哪些field

GET /test\_index/test\_type/1?\_source=test\_field1,test\_field2

{

"\_index": "test\_index",

"\_type": "test\_type",

"\_id": "1",

"\_version": 2,

"found": true,

"\_source": {

"test\_field2": "test field2"

}

}

## document 的全量替换、强制创建以及图解 lazy delete 机制

1、document的全量替换

2、document的强制创建

3、document的删除

**1、document的全量替换**

（1）语法与创建文档是一样的，如果document id不存在，那么就是创建；如果document id已经存在，那么就是全量替换操作，替换document的json串内容

（2）document是不可变的，如果要修改document的内容，第一种方式就是全量替换，直接对document重新建立索引，替换里面所有的内容

（3）es会将老的document标记为deleted，然后新增我们给定的一个document，当我们创建越来越多的document的时候，es会在适当的时机在后台自动删除标记为deleted的document

**2、document的强制创建**

（1）创建文档与全量替换的语法是一样的，有时我们只是想新建文档，不想替换文档，如果强制进行创建呢？

（2）PUT /index/type/id?op\_type=create，PUT /index/type/id/\_create

**3、document的删除**

（1）DELETE /index/type/id

（2）不会理解物理删除，只会将其标记为deleted，当数据越来越多的时候，在后台自动删除



## 深度剖析 Elasticsearch 并发冲突问题



## 深度图解剖析悲观锁与乐观锁两种并发控制方案

## 图解 Elasticsearch 内部如何基于\_version 进行乐观锁并发控制

（1）\_version元数据

PUT /test\_index/test\_type/6

{

"test\_field": "test test"

}

{

"\_index": "test\_index",

"\_type": "test\_type",

"\_id": "6",

"\_version": 1,

"result": "created",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

},

"created": true

}

第一次创建一个document的时候，它的\_version内部版本号就是1；以后，每次对这个document执行修改或者删除操作，都会对这个\_version版本号自动加1；哪怕是删除，也会对这条数据的版本号加1

{

"found": true,

"\_index": "test\_index",

"\_type": "test\_type",

"\_id": "6",

"\_version": 4,

"result": "deleted",

"\_shards": {

"total": 2,

"successful": 1,

"failed": 0

}

}

我们会发现，在删除一个document之后，可以从一个侧面证明，它不是立即物理删除掉的，因为它的一些版本号等信息还是保留着的。先删除一条document，再重新创建这条document，其实会在delete version基础之上，再把version号加1



## 上机动手实战基于\_version 进行乐观锁并发控制

测试内容